Kevin Barbour

922119026

10/20/2021

Assignment 3 Documentation

Github Repository

https://github.com/KevBarbour15/assignment-3-parser-KevBarbour15-assignment-3.git

Project Intro and Overview

Assignment 3 moves from our last assignment of the Lexer on to using the Parser part of our codebase. In it, it is required that we expand the production rules within the Parser to expand the grammar used in our X language. Additionally, we are required to create a Visitor that neatly draws out a tree of our X language.

Scope of Work

Key:

X = Successfully completed

X = partially completed/final result differs from requirement

Requirement 1: Modify the Parser.java file to be able to handle new grammar with the necessary production rules, including update the tokens file with the new grammar. New production rules are:

TYPE —> ‘number’ **X**

TYPE —> ‘date’ **X**

F —> <number> **X**

F —> <date> **X**

S —> ‘if’ E ‘then’ BLOCK **X**

S —> ‘doloop’ BLOCK ‘until’ E **X**

S —> ‘for’ NAME ‘in’ LIST BLOCK ‘else’ BLOCK **X**

LIST —> ‘[‘F’,’F’]’ **X**

E —> SE ‘>’ SE **X**

E —> SE ‘>=’ SE **X**

For the requirements above, I created new classes under the ast file to accommodate the new grammar. The new classes are **DateLitTree.java, DateTypeTree.java, DoTree.java, ForTree.java, ListTree.java, NumberLitTree.java** and **NumberTypeTree.java.** To implement the new trees, I created the aforementioned new classes, as well as followed the patterns I saw already in the Parser.java class to correctly follow the productions required. To implement the ‘if’ statement without an ‘else’ I added conditional code for if an ‘else’ is read in.

To implement the List, I created an rList() method. I struggled with getting it to work correctly, so in the interest of neatness and final execution (as well as being needed to implement the ‘for’ grammar, the method accepts a LeftBracket token, an integer, a comma, another integer and a right bracket.

Requirement 2: Remove all of the debugger print out statements that were part of the program provided. **X**

Removing the debugger print statements required me to trace where the print statements were coming from and delete them, as well as delete the countless debugger statements I use through out development.

Requirement 3: Create 2 new visitors, OffsetVisitor.java and DrawOffsetVisitor.java. **X**

- **OffsetVisitor.java**: In OffsetVisitor, I implemented HashMaps labeled **offsets** to keep track of the offsets of each node in the tree. I created a method titled **getMaxOffset()** to keep track of the maximum offset for width purposes when drawing the offsets. I created a method called **getParentOffset()** that gets the offset of the leftmost and rightmost child and does simple arithmetic to return the correct parent offset to be used in the algorithm. The method **setOffset()** is used for my attempt to implement the algorithm provided in the assignment documentation. I went through many iterations and attempts at implementing the algorithm including

using a helper method for the recursive part, but ultimately I was unsuccessful in getting the nodes to sit in the correct offset before the deadline. I opted to remove the helper method to adjust the kids in the interest of code cleanliness. The resulting tree drawing is not correct due to the offsets being incorrect. I am not satisfied with my final **setOffset()** method, as I felt I was very close to figuring it out throughout the development process, but because of tinkering with it until the end I had to ensure that the code did not break when run and at minimum produced an image before the deadline.

- **DrawOffsetVisitor.java**: For this class, I mirrored the **DrawVisitor.java** class provided, while making some changes to accommodate the drawing the offsets. It takes in the nCount[] array, as well as the offsets HashMap and the int maxOS as parameters. The maxOS is the maximum offset to be used for setting the width of the image, and offsets is used to passed the offset position in to be drawn. Additionally, I had to update where the line connecting the nodes is drawn to show the flow of the tree.

Execution and Development Environment

Developed Assignment 3 on VS Code on my MacBook Pro. I used the terminal in VS Code as well as the terminal on the MacBook to test my program.

Compilation Result

There were no compilation issues, and my project opens and runs correctly. I was able to successfully run java compiler.Compiler with the samples files provided, as well as numerous ones I created to test the new productions.

Assumptions

There are no assumptions for my project.

Implementation

For my implementation of the new productions for the grammars, I was able to pick up on the patterns for the grammars already in place. For the implementation of the visitors, I attempted to follow the algorithm as described in the assignment handout. I went through many iterations and attempts of it unsuccessfully. Attempting to include my own recursive methods proved to be extremely difficult for me. In order to keep the code clean, I used my best judgement for whitespace usage and deleted the countless

comments and print statements I include as I debug.

Results and Conclusion

My program runs mostly correct. However, the most noticeable and potentially largest part of the assignment is noticeably incorrect. I have found this (and the last) project to be extremely rewarding, and of course at many times frustrating. I enjoy how large the program provided is. Some of the concepts, like the Visitors, proved to be very confusing. That combined with the recursive portions really blew my mind at times, but I believe I can figure any of it out with more work.

Future Work

I look forward to more projects of this size. Things I want to focus on are studying up/refreshing recursive methods because those can be a huge obstacle without the right understanding. Additionally, I want to get better and more confident in asking for assistance when I am stuck. I tend to spin my wheels in place and end up stuck on a specific issue, as opposed to seeking help and moving on.